

Enabling Fast, Noncontiguous GPU Data Movement in Hybrid MPI+GPU Environments

John Jenkins, Nagiza Samatova

– North Carolina State University

James Dinan, Pavan Balaji,
Rajeev Thakur

– Argonne National Laboratory

Contact: Pavan Balaji (balaji@mcs.anl.gov)

Overview

- MPI-ACC
- Contributions
- GPU, MPI, datatypes background
- Datatype processing algorithm
- Experimental evaluation



Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

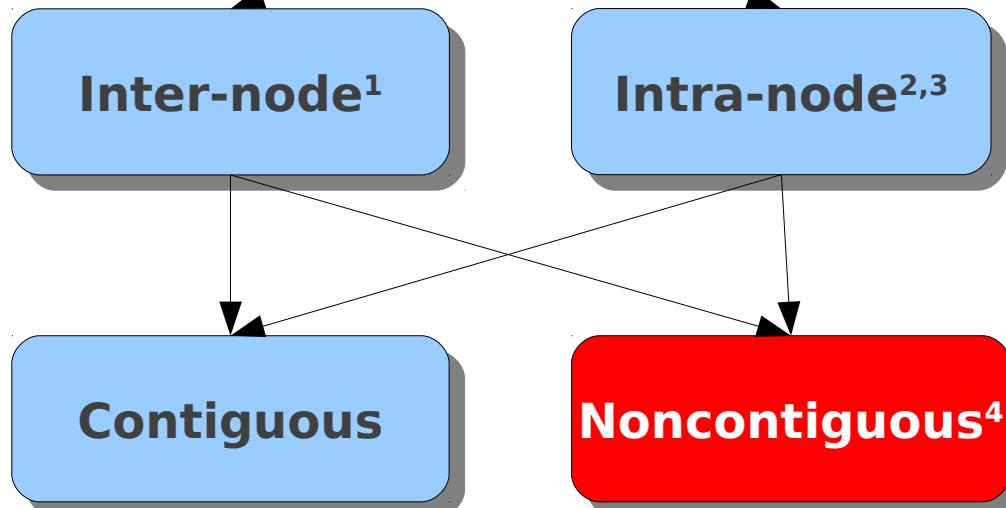
MPI-ACC

Accelerator-aware Data Movement Within MPI

Multiple accel.
(OpenCL and CUDA)

Portably leverage system
and vendor-specific
optimizations

- Attribute-driven
- NUMA-aff. aware
- PCIe-aff. aware
- OpenCL handle caching



- I/O Hub Aware,
- IPC handle reuse,
- Low overhead

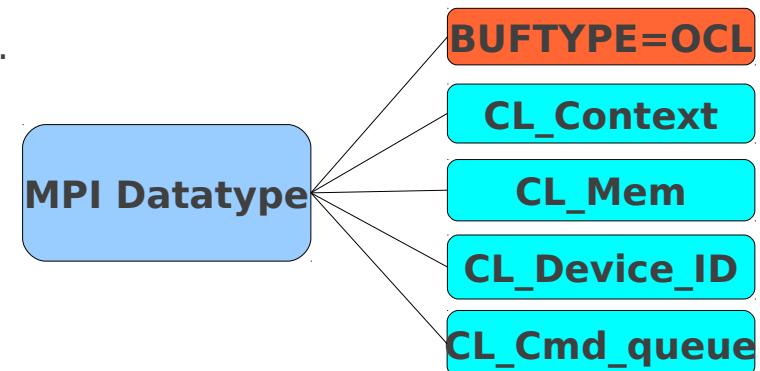
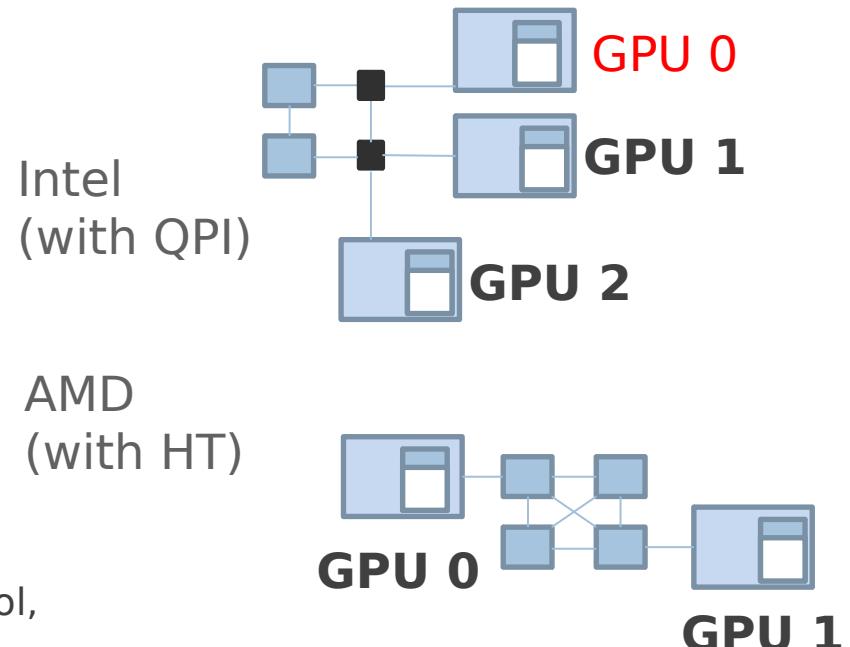
- 1.** Aji et al. *MPI-ACC: An Integrated and Extensible Approach to Data Movement in Accelerator-Based Systems*. HPCC '12.
- 2.** Ji et al. *DMA-Assisted, Intranode Communication in GPU Accelerated Systems*. HPCC '12.
- 3.** Ji et al. *Efficient Intranode Communication in GPU-Accelerated Systems*. ASHES '12 Workshop.
- 4.** Jenkins et al. *Enabling Noncontiguous GPU Data Movement in Hybrid MPI+GPU Environments*. Cluster '12.



Contact: Pavan Balaji (balaji@mcs.anl.gov)

MPI-ACC – Features

- Communication
 - Dynamic Pipelining
 - NUMA-affinity aware
 - PCIe-affinity aware
 - Topology-agnostic
 - Extended large-message protocol, handle passing
- API
 - Explicit Interfaces
 - MPI_CUDA_Send, MPI_OpenCL_Recv, etc.
 - Datatype attributes
 - Elegantly handles different accelerator views of memory
 - E.g. CUDA void* vs. OpenCL context, memory handles.



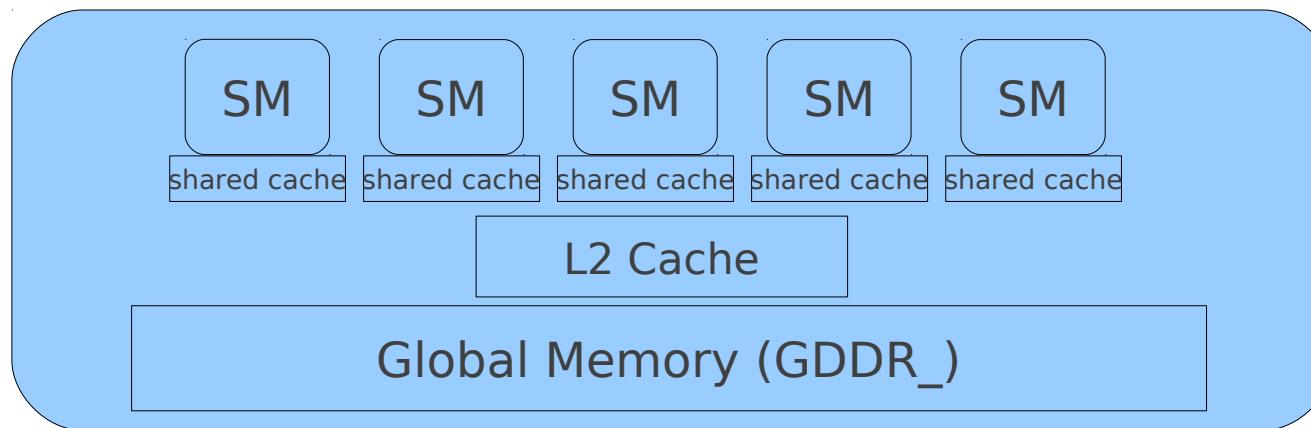
Contributions

- MPI datatype processing algorithm:
 - generalized to **any** datatype
 - fine-grained parallel, GPU-optimized
- Investigate resource contention scenarios for packing on the GPU, for both DMA-based and kernel-based packing.



Contact: Pavan Balaji (balaji@mcs.anl.gov)

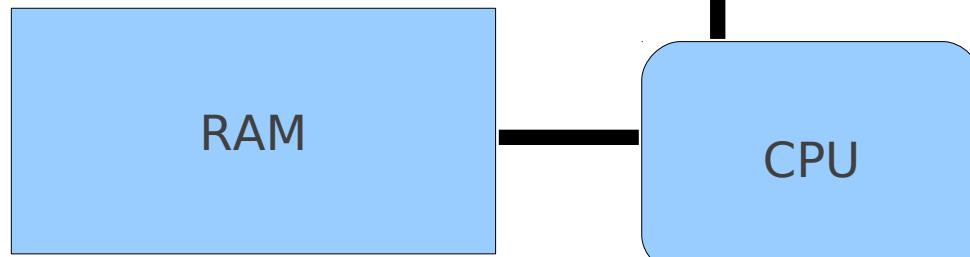
GPU Memory Spaces / Interconnect



PCI Bus
- High Bandwidth
- High Latency

PCI Controller

NIC



**GPU/CPU
Memory
Distinct!**

Contact: Pavan Balaji (balaji@mcs.anl.gov)



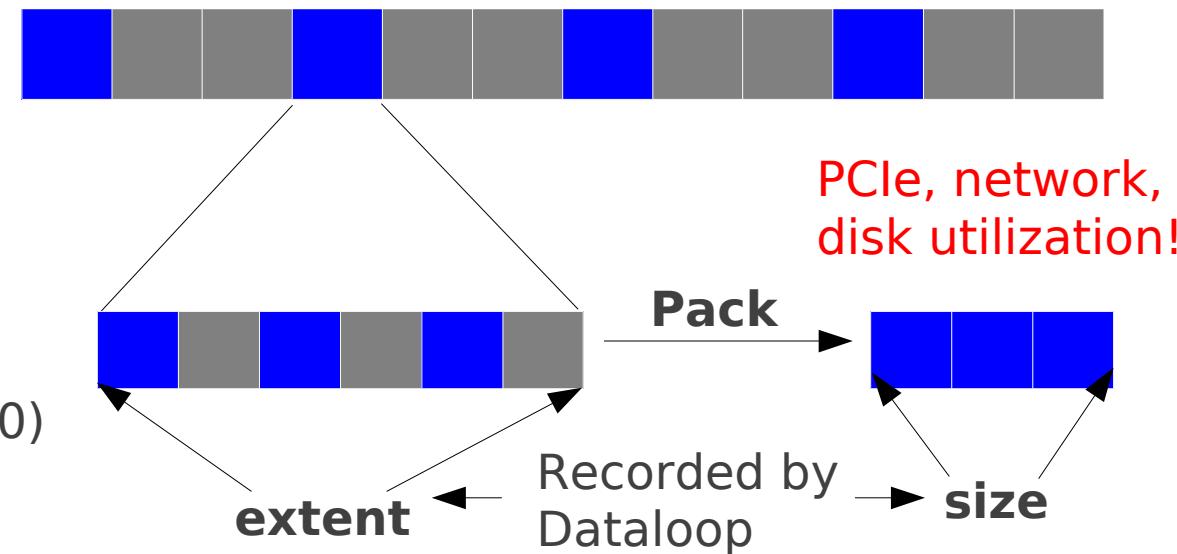
NC STATE UNIVERSITY

Department of Computer Science

Derived Datatypes: Vector Example

- Communication, I/O on noncontiguous data
- MPI_Type_vector(count, blocklength, stride, oldtype, newtype)
Parameters:
 - count – number of blocks
 - blocklength – number of contiguous oldtype per block
 - stride – distance between elements (wrt oldtype or bytes)

`MPI_Type_vector(
 4, 1, 3, v0, v1)`



`MPI_Type_vector(
 3, 1, 2, DOUBLE, v0)`



GPU Datatype Processing Goals

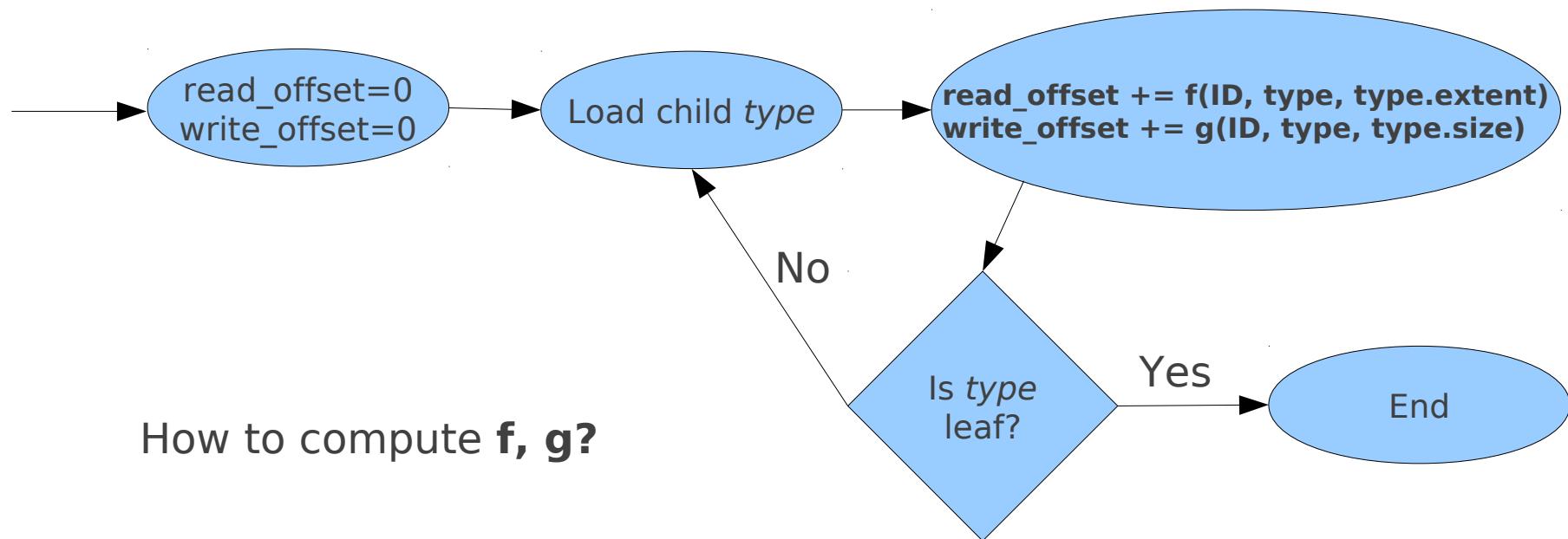
Characteristic	Reason	CPU processing comparison
Pack non-contiguous data	Efficient PCIe, network, disk usage	Same
Fine-grain parallel	GPU arch.	Serial
No inter-thread dependencies	Avoid parallel computation stalls	Stack-based packing state
Computation/transfer overlap	PCIe, network best practice	Same, but no PCIe
Compact datatype representation	Multiprocessor caching, memory optimization	Tree-based



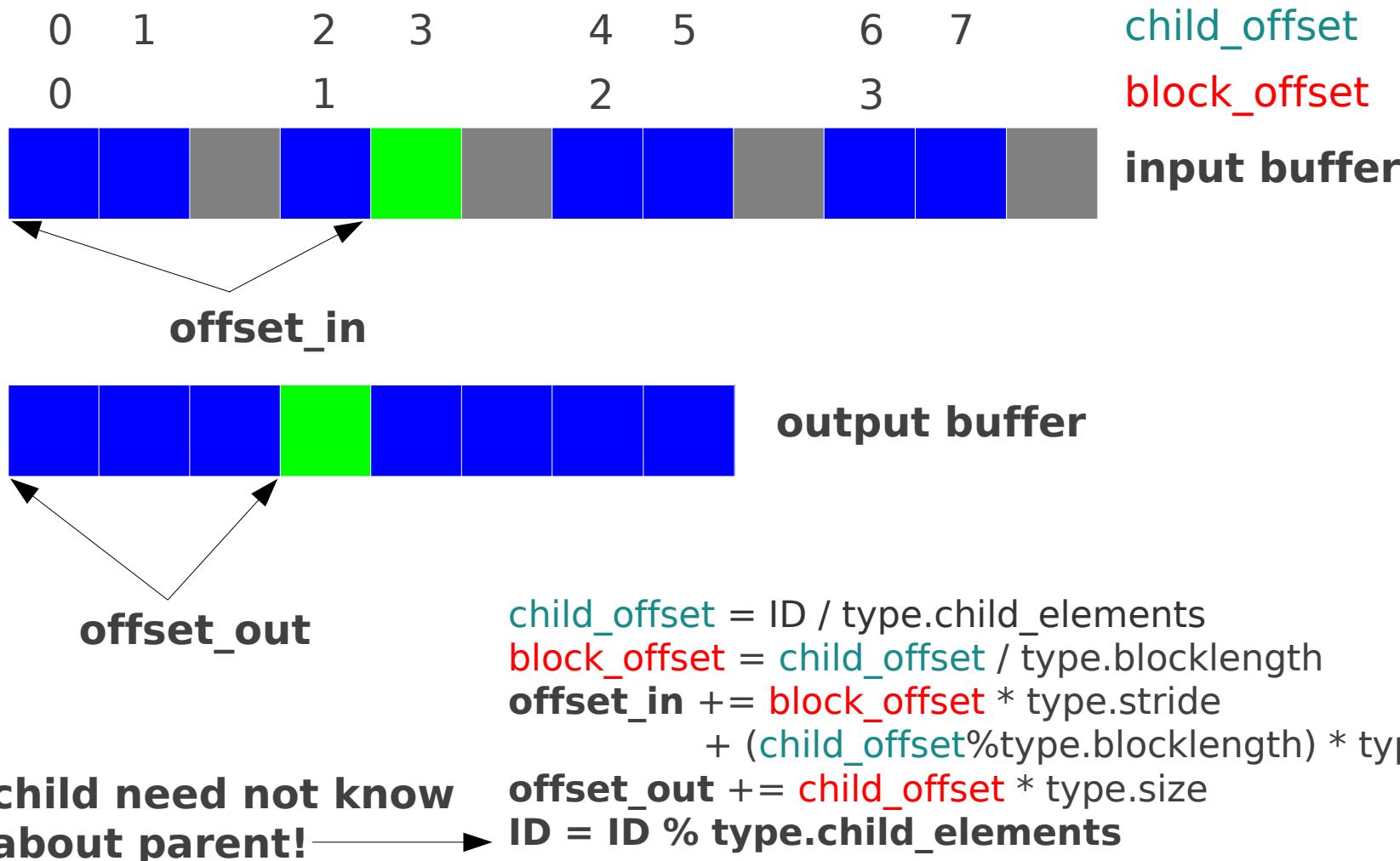
Contact: Pavan Balaji (balaji@mcs.anl.gov)

Traversal Algorithm

- Key Insight – We can determine where each element (double, int, etc.) is using only datatype encoding + number of primitive elements per datatype!
- input: element ID output: read_offset, write_offset

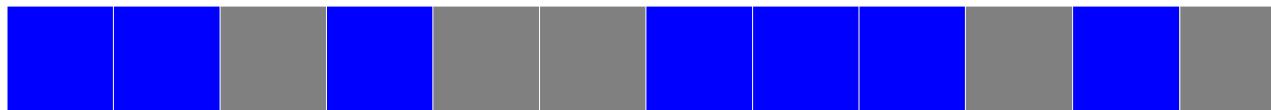


Example Offset Computation: Vector



Contact: Pavan Balaji (balaji@mcs.anl.gov)

Variable length handling: Blockwise binary search (indexed, struct)



blocklength	2	1	3	1
prefix sum	0	2	3	6

(7 threads)

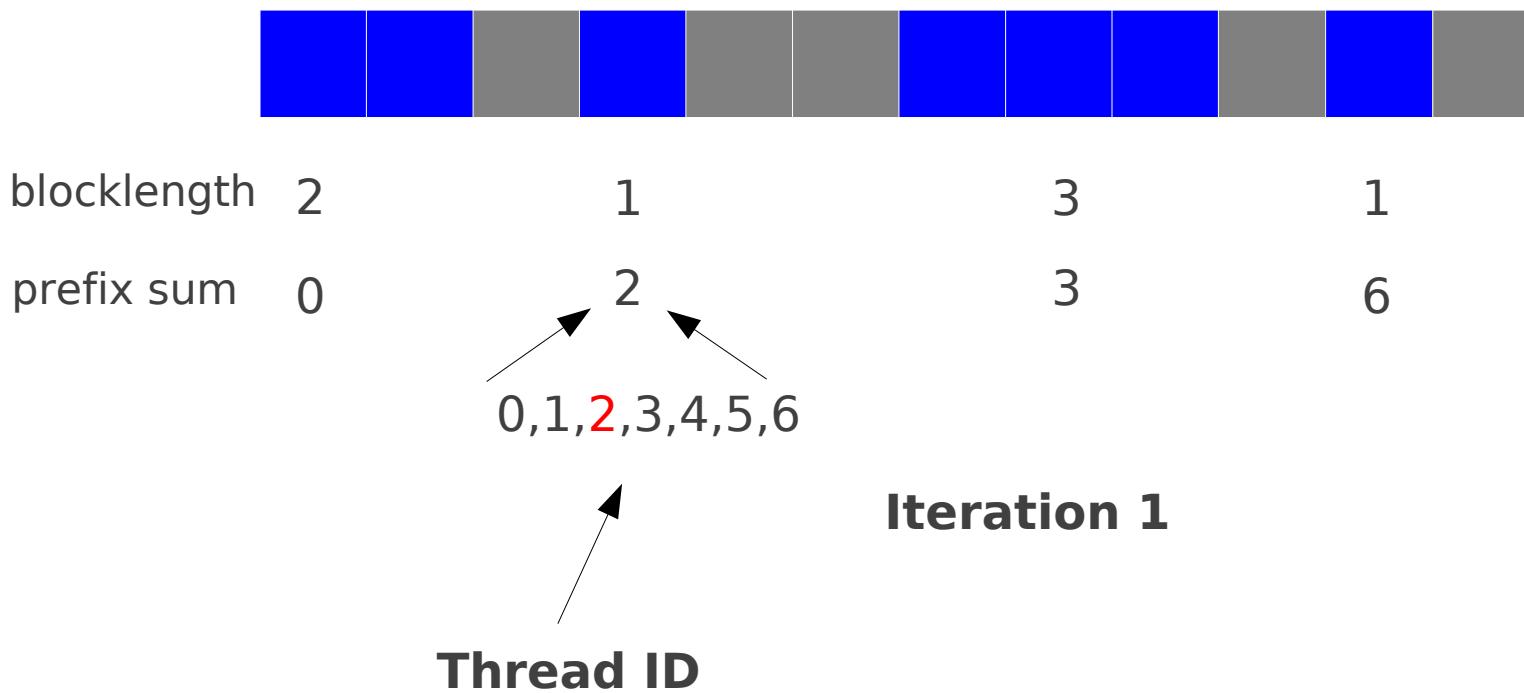


Contact: Pavan Balaji (balaji@mcs.anl.gov)

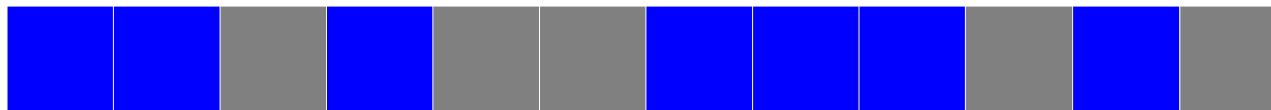
NC STATE UNIVERSITY

Department of Computer Science

Variable length handling: Blockwise binary search (indexed, struct)



Variable length handling: Blockwise binary search (indexed, struct)



blocklength	2	1	3	1
-------------	---	---	---	---

prefix sum	0	2	3	6
------------	---	---	---	---

0,1

3,4,5,6

Iteration 2

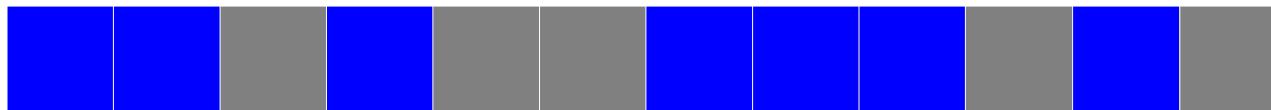


Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

Variable length handling: Blockwise binary search (indexed, struct)



blocklength	2	1	3	1
prefix sum	0	2	3	6

↑
6

Iteration 3



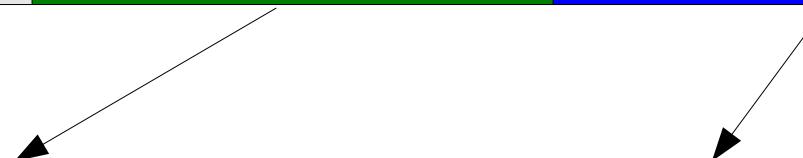
Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

GPU Datatypes Representation

Type	Fixed	Variable
Common	count, size, extent, #primitives	
Contig		
Vector	stride, blocklength	
Indexed	lookaside offset	blocklengths, displacements
Blockindexed	blocklength, lookaside offset	displacements
Struct	lookaside offset	blocklengths, displacements, types, child type IDs
Subarray	dimensions, lookaside offset	start offsets, sizes, subsizes



Inorder Buffer

cache in shared memory

Lookaside Buffer

cache if enough space



Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

Algorithm Characteristics - Summary

- Cache
 - Fixed-size parameter caching
 - Variable-length parameter caching if size permits
- Coalescence
 - Type representation load – fully coalesced
 - High on output – contiguous data
 - Implicit on input (adjacent threads, adjacent items)
 - Sensitive to data distribution
- Thread Branch Divergence
 - Only on indexed, struct types
 - Clustering due to locality creates similar search paths
- Bus Efficiency
 - Single, contiguous writes → zero-copy!



Contact: Pavan Balaji (balaji@mcs.anl.gov)

Benchmarking - Test Datatypes

Datatype	CUDA Impl.	Notes
contiguous	cudaMemcpy	Kernel overhead.
2D-vector*	cudaMemcpy2D	Most common. CUDA perf. sensitive to data layout
4D-subarray	Iterative cudaMemcpy3D	Common. Cannot be represented by vector type.
indexed	cudaMemcpy per-block	Irregular, does not map well to CUDA DMA.
blockindexed	cudaMemcpy per-block	No binary search, compare against indexed
“C-style” struct	cudaMemcpy per-extent	Branch divergence on read/write.

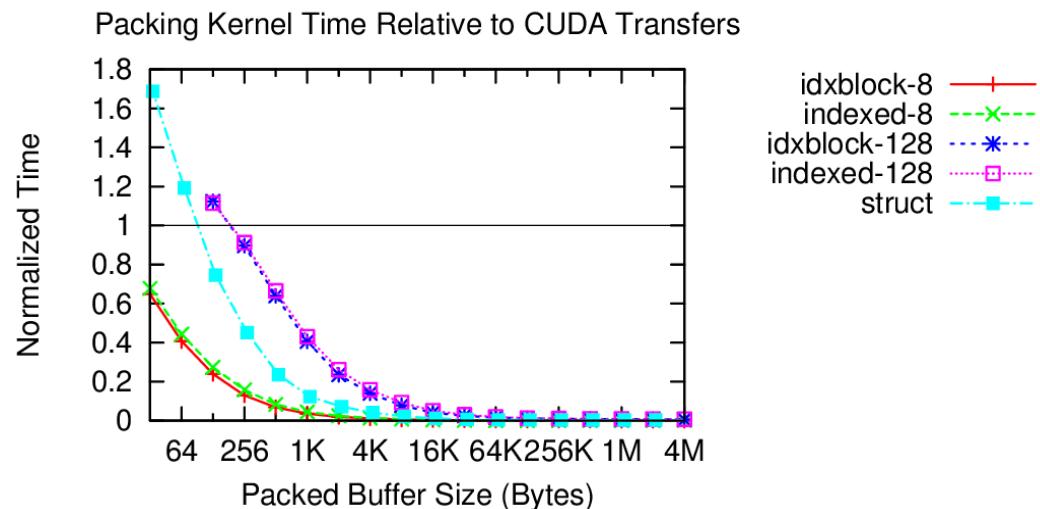
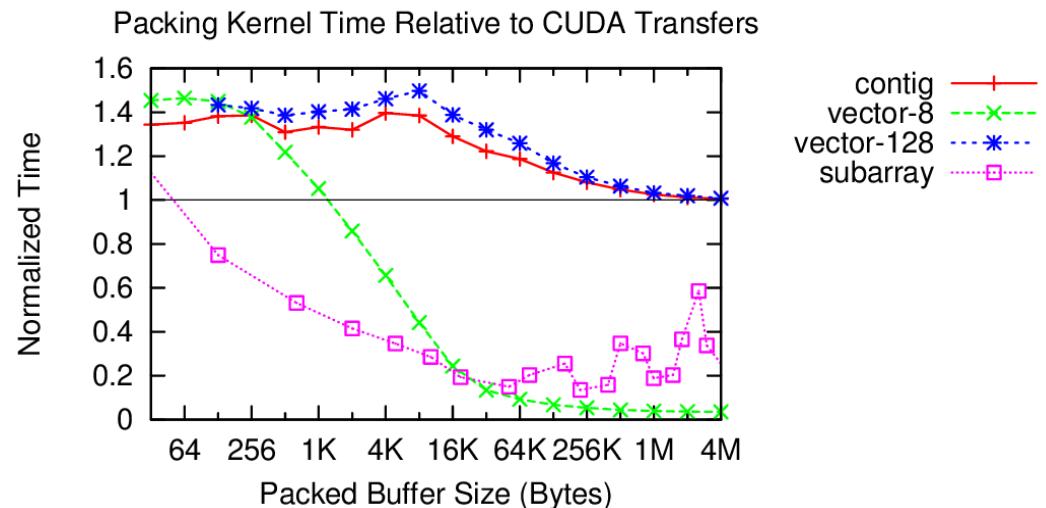
* Aligned on CUDA-optimized byte boundaries



Contact: Pavan Balaji (balaji@mcs.anl.gov)

Comparison – CUDA DMA

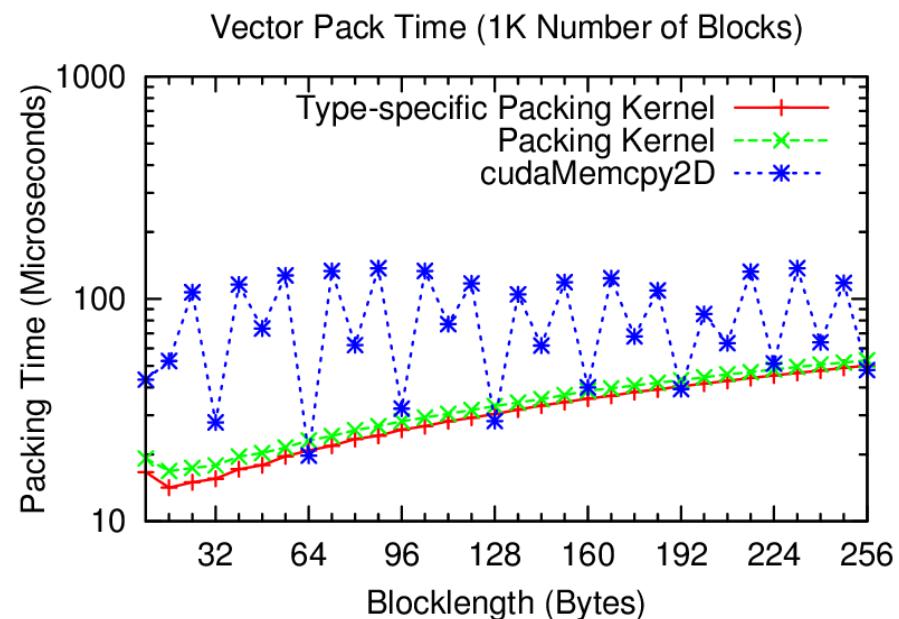
- **Vector** – depends on parameterization (later)
- **Subarray** – latency aggregated in CUDA calls.
- **Irregular types** – huge speedup (no reasonable CUDA equivalent)
- **Overhead** – \sim a few μs



Contact: Pavan Balaji (balaji@mcs.anl.gov)

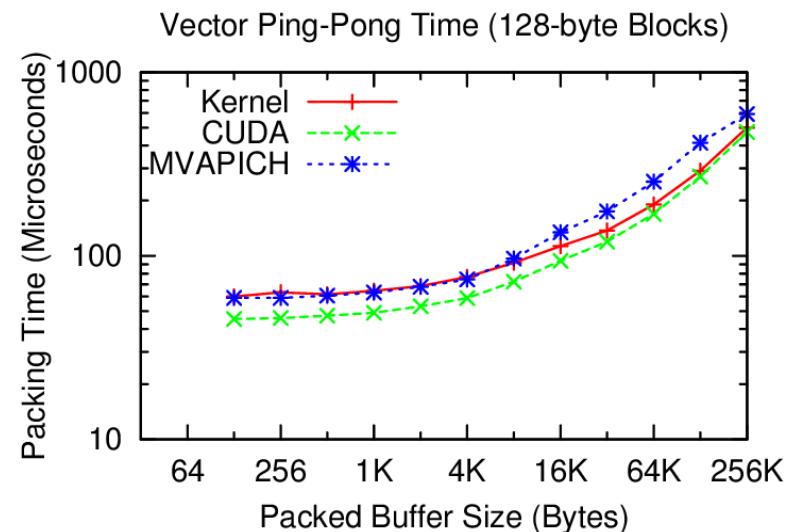
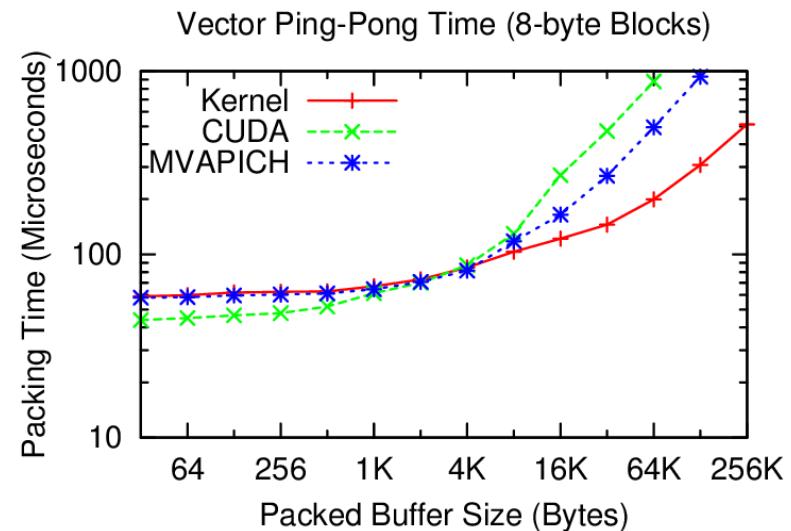
Comparison – Vector Parameterizations

- CUDA performs best on multiples of 64 bytes.
- CUDA performs poorly otherwise.
- Same goes for vector stride.

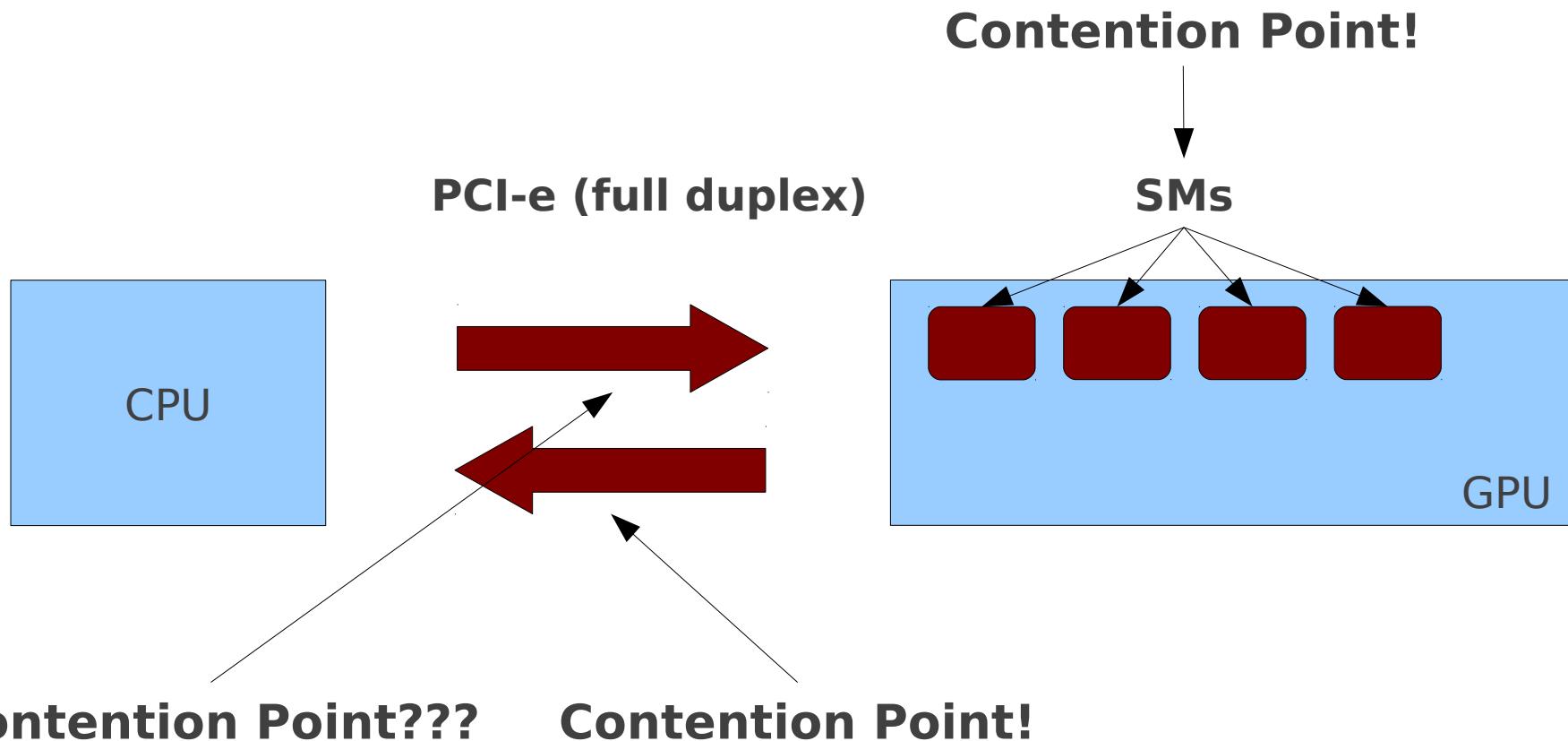


Comparison – Vector Communication (Ping Pong)

- **CUDA DMA** best for small buffers, large blocks.
 - **NOTE:** data laid out to be CUDA-optimal
- **Packing kernel** best for larger buffers, small blocks.
- **MVAPICH** – serialized packing, PCIe (ours fully pipelined through zero-copy).
 - Performance doesn't carry over to other datatypes – tied to CUDA DMA.



Resource Contention



Contact: Pavan Balaji (balaji@mcs.anl.gov)

Resource Contention

	SM	CPU → GPU	GPU → CPU
Packing (kernel)	yes	somewhat*	yes**
CUDA	no	somewhat*	yes

* shouldn't happen (scheduler artifact?)

** PCIe transactions driven by SMs (zero copy) treated more favorably



Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

Results Summary

- ✓ Use packing for
 - ✓ irregular types (**Order of Magnitude Speedup**)
 - ✓ large sparse transfers (**Order of Magnitude Speedup**)
 - ✓ types which do not adhere to CUDA-optimized memory layouts
- Use CUDA DMA for
 - small transfers,
 - 2D, 3D arrays with large, contiguous chunks.
- Use hand-coded packing kernels for small sized, simple types.

- ✓ **Datatypes implementations can control for these cases!**
 - Packing is **complementary** rather than competing



Contact: Pavan Balaji (balaji@mcs.anl.gov)

NC STATE UNIVERSITY

Department of Computer Science

Thanks!

Acknowledgements:

- U.S. Department of Energy under contract DE-AC02-06CH11357
- National Science Foundation under Grant No. 0958311.

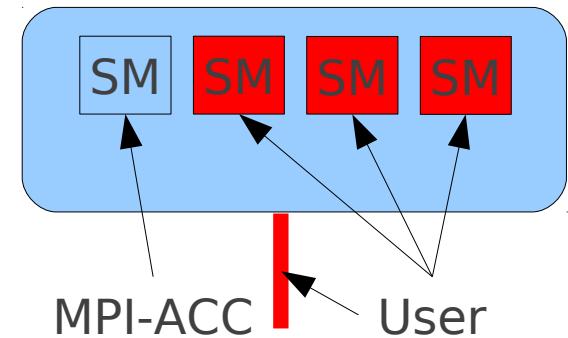
Contact: Pavan Balaji (balaji@mcs.anl.gov)

Extra Slides

Contact: Pavan Balaji (balaji@mcs.anl.gov)

Future Work / CUDA “Wish List”

- Decide which packing method best at runtime (CUDA DMA, kernel, etc.)
- We don't have control over what's going on in the GPU
 - Greedy scheduler
 - Priority mechanisms?
- CUDA, OpenCL can't query GPU utilization!
 - Nvidia-smi – coarse grain measurement
 - Need queue information!
- Other ideas
 - persistent kernel (allocate single SM for entirety of MPI application)



Contact: Pavan Balaji (balaji@mcs.anl.gov)

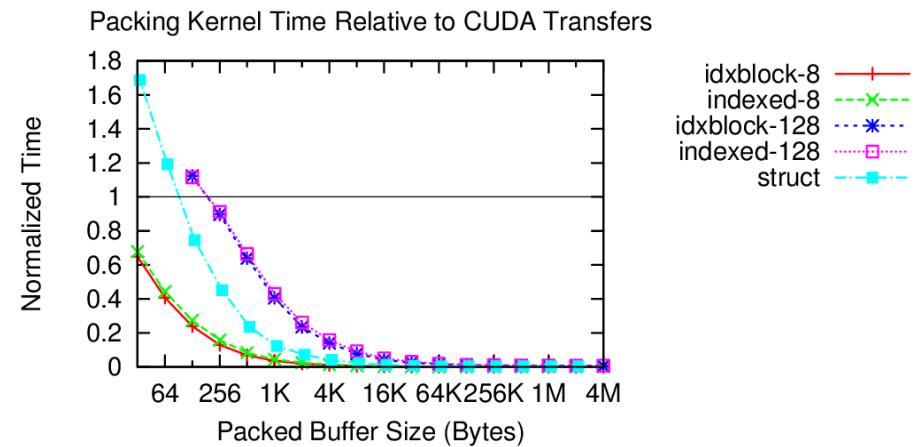
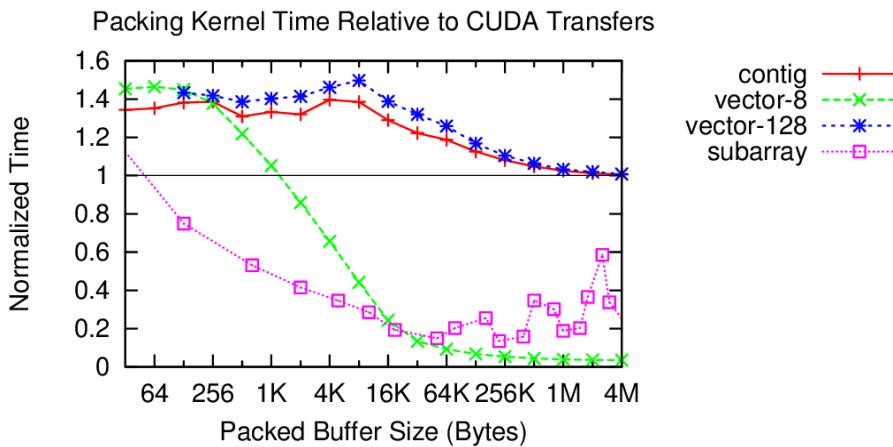
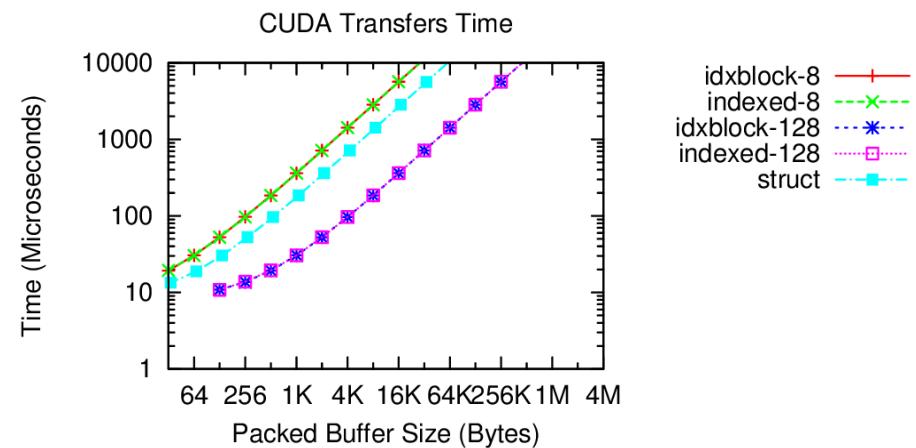
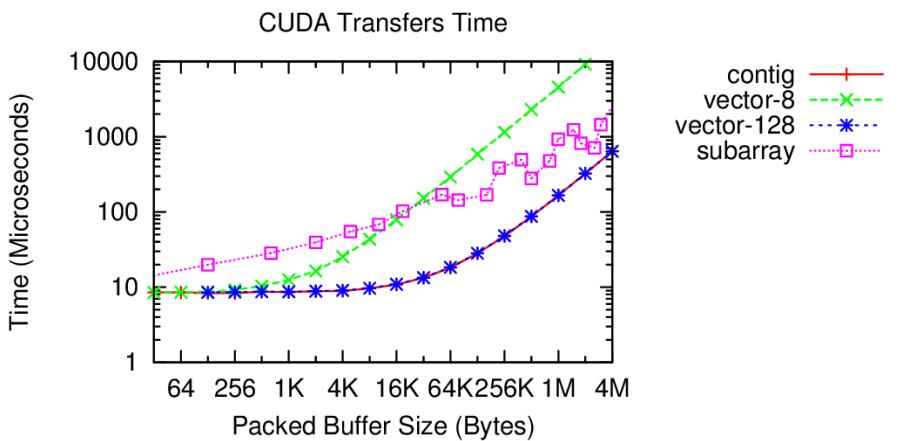
MPI+GPU Support in Other Libraries

- MVAPICH (Wang et al., ISC '11, Cluster '11)
 - Allows communication of contiguous CUDA GPU buffers, vector datatypes using 2D DMA, derived datatypes using DMA per item.
 - Enabled through CUDA Unified Virtual Addressing (UVA).
- OpenMPI
 - Enabled through CUDA UVA, derived datatypes using DMA per item.
- DCGN (Stuart & Owens, IPDPS '09)
 - GPU-sourced communication, using polling CPU thread.

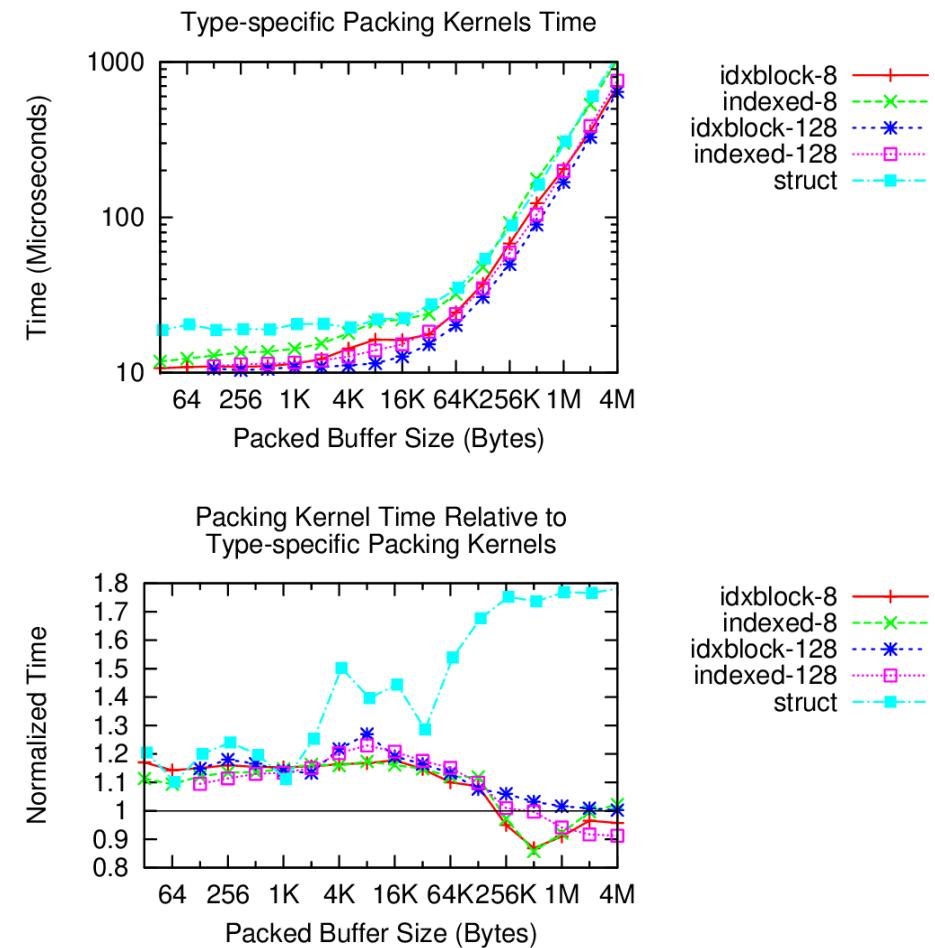
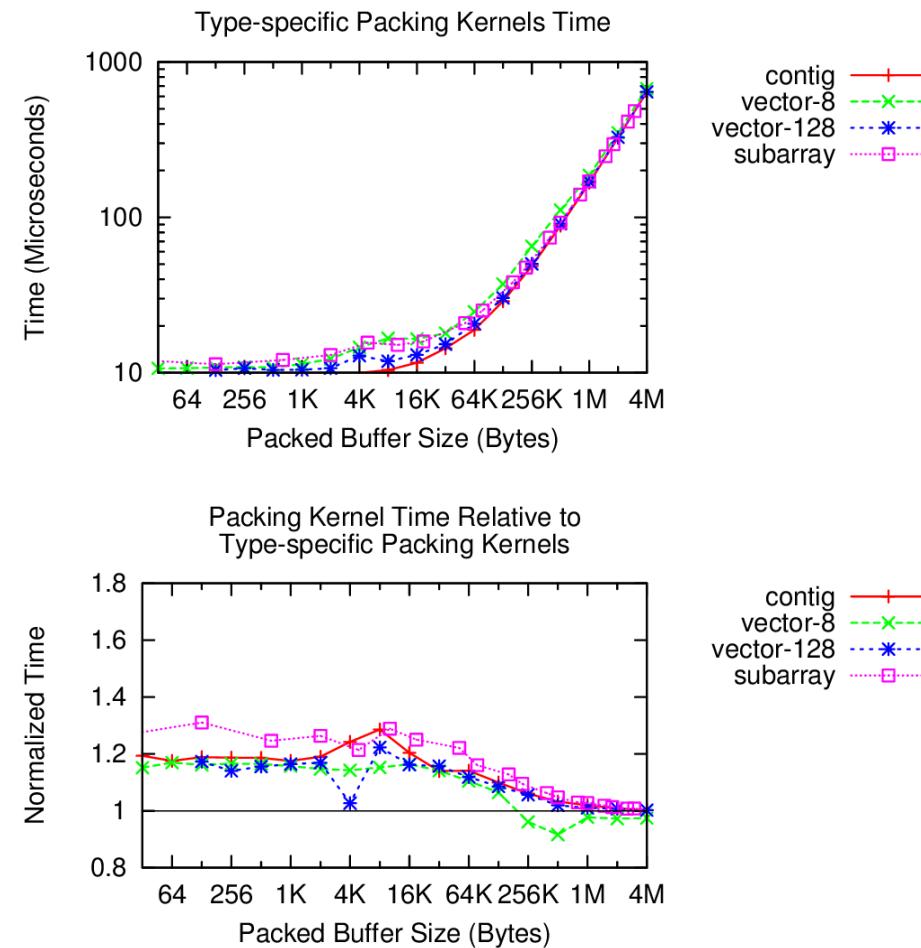


Contact: Pavan Balaji (balaji@mcs.anl.gov)

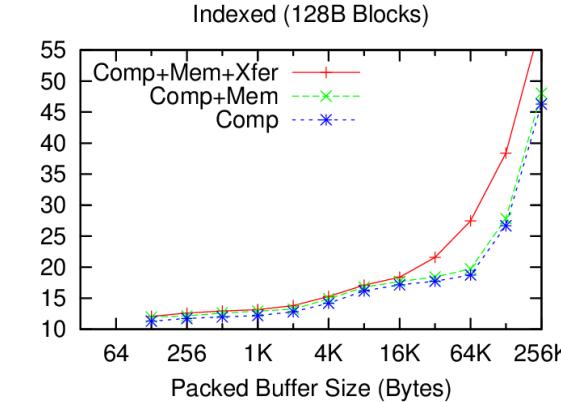
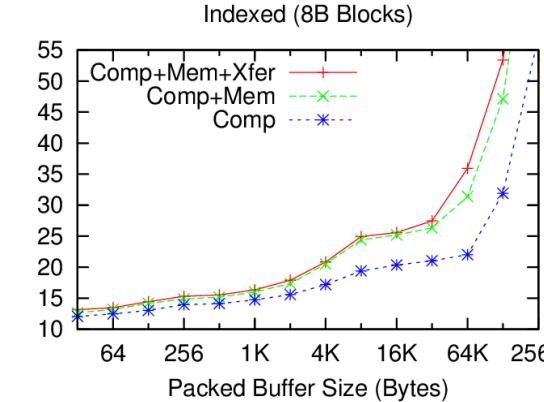
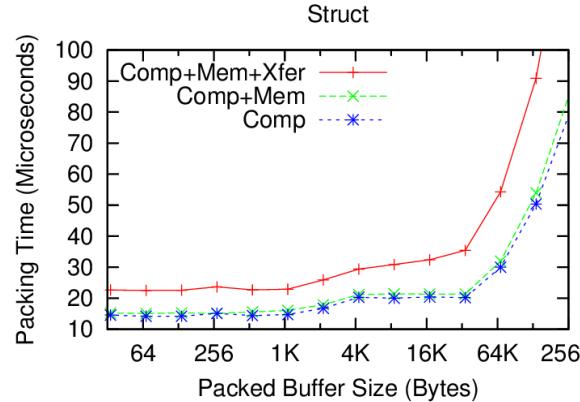
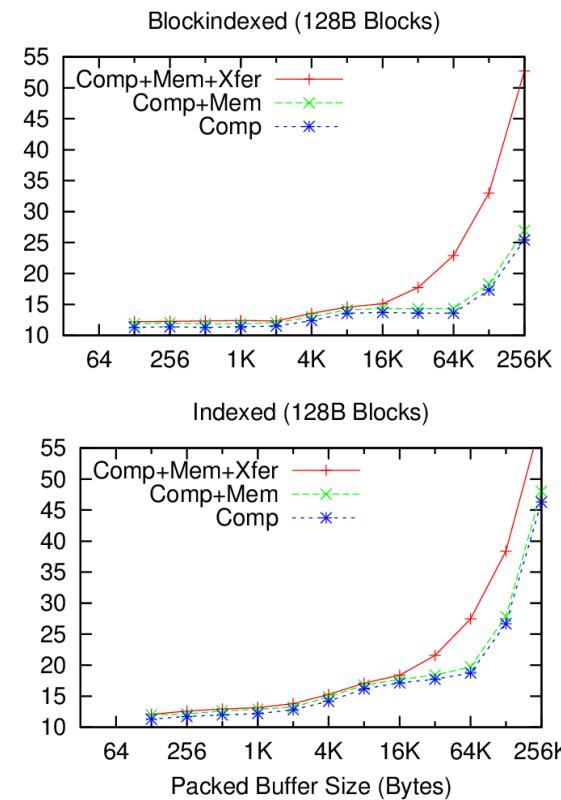
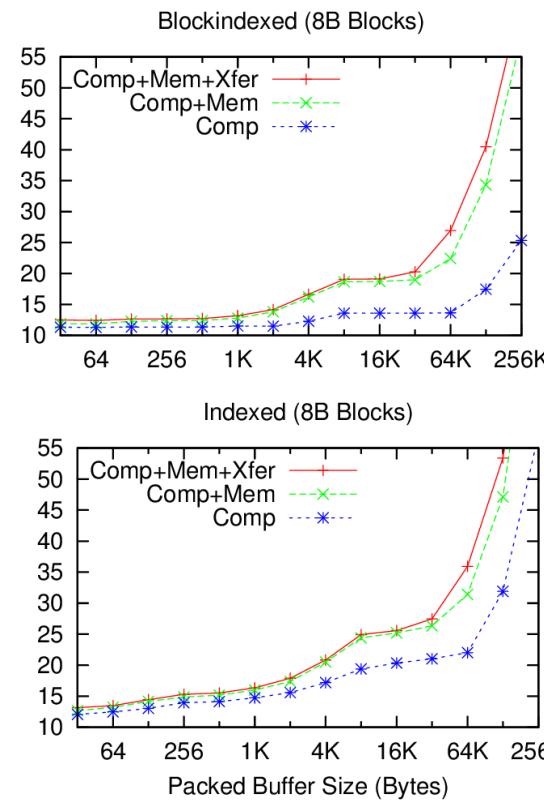
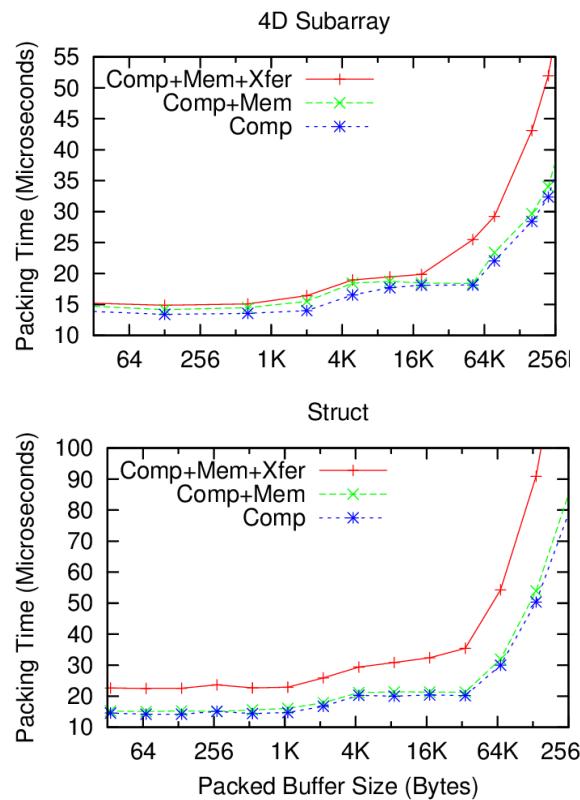
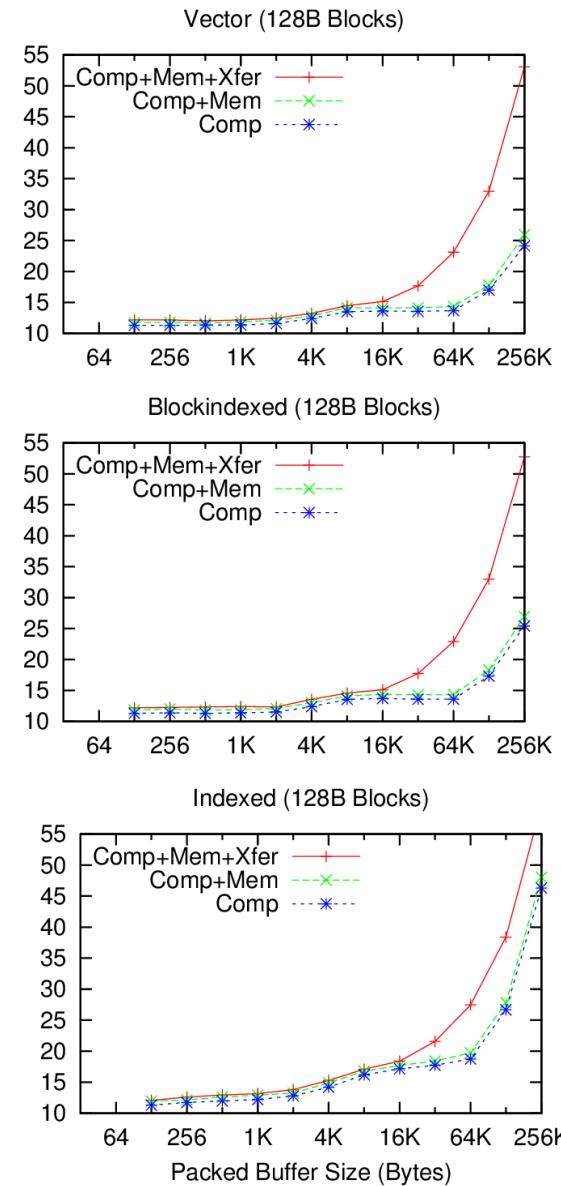
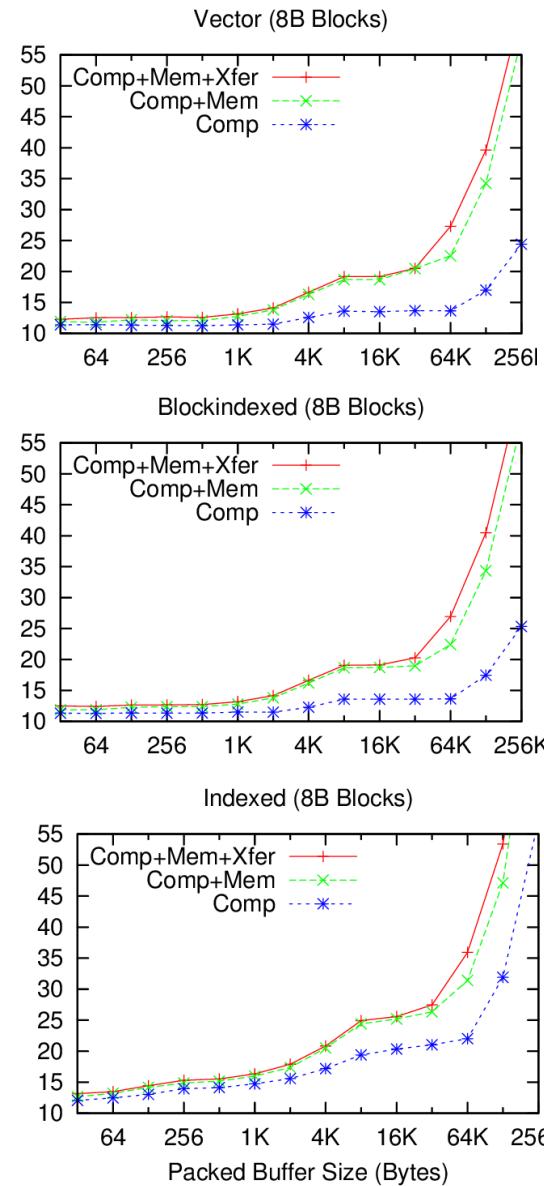
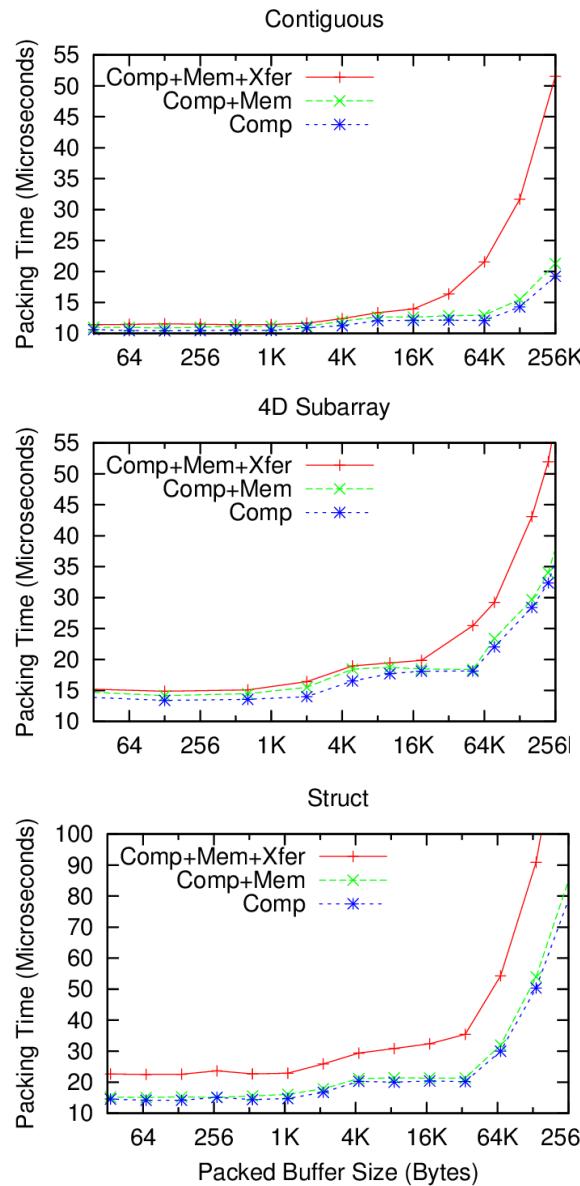
Comparison – CUDA DMA



Comparison – Type-specific Packing Kernels



Comparison – Packing Component Costs



Contact: Pavan Balaji (balaji@mcs.anl.gov)



Comparison – Vector Communication

